ARI Research Note 87-45

A Programmer's Guide to the Sensory-Effector Interface

Keith Barnett
Carnegie-Mellon University

for

Contracting Officer's Representative
Judith Orasanu

BASIC RESEARCH LABORATORY
Michael Kaplan, Director

U. S. Army
Research Institute for the Behavioral and Social Sciences

October 1987

U. S. ARMY RESEARCH INSTITUTE
FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency under the Jurisdiction of the

Deputy Chief of Staff for Personnel

EDGAR M. JOHNSON
Technical Director

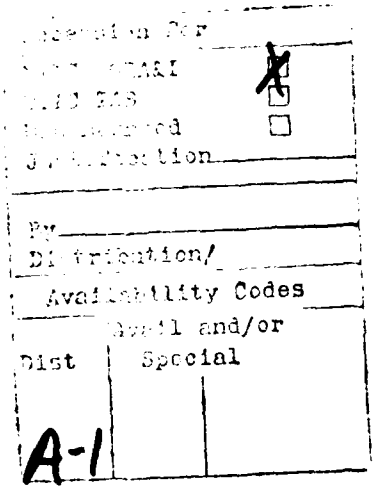
WM. DARRYL HENDERSON
COL, IN
Commanding

Research accomplished under contract
for the Department of the Army

Carnegie-Mellon University

Technical review by

Dan Ragland



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ARI Research Note 87-45	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* A Programmer's Guide to the Sensory-Effector Interface		5. TYPE OF REPORT & PERIOD COVERED Interim Report January 86 - January 87
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Keith Barnett		8. CONTRACT OR GRANT NUMBER(s) MDA903-85-C-0324
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department Carnegie-Mellon University Pittsburgh, PA 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 2Q161102B74F
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Research Institute for the Behavioral and Social Sciences, 5001 Eisenhower Avenue, Alexandria, VA 22333-5600		12. REPORT DATE October 1987
		13. NUMBER OF PAGES 8
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)* - -		15. SECURITY CLASS. *(of this report)* Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE n/a

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

- -

18. SUPPLEMENTARY NOTES

Judith Orasanu, contracting officer's representative

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Sensory-Effector Interface (SEI)
Artificial Intelligence
World Modeling System
World Master
Computer Models

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This document describes the Sensory-Effector Interface (SEI) provided as part of the World Modeling System, and explains how to build a new SEI or extend the existing one.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

Table of Contents

1. Introduction

An SEI performs three major functions: maintaining its own copy of the world data structures, selecting information from these structures appropriate to the perception capabilities of a particular organism, and translating actions taken by an organism into changes in the world data structures.

An organism using a world communicates with it only indirectly, through an SEI; each organism using a world will have its own SEI. Since an organism may be written in any language, there will exist many versions of the SEI; a core of C routines allowing an SEI to communicate with a world, however, should be common to all SEIs.

In addition to a connection to its organism (which may be part of the same process) an SEI maintains a connection to the World Master, from which it receives updates of the world data structures; only portions of this information will be directly observable by the organism. An SEI may also have a connection to a graphics process, from which it may receive pixel values or a list of objects visible to its organism.

See *The World-Wide Communication Protocol* and *The World Data Structure*.

2. Message Handling

2.1 Connecting to the World

As described in the protocol, the SEI must connect to the World Coordinator and request a unique ID, which, once received, forms part of the name the SEI checks in for its port. It will receive a message from the World Master, to which it must respond with its ID. The master will also indicate the name of the organism for which the SEI is responsible, and the object in the world to which it corresponds. The SEI must must wait for an initial copy of the world, and may then function normally until forced to exit by the master or its organism.

A connection to a graphics process is not explicitly made in the SEI. Use of graphics is specified upon startup or by request from the organism; the graphics process will make the necessary connection, and send the appropriate information. When using vision in this way, the SEI can either wait to service its organism until it has received all information from both the master and graphics processes, or (since the graphics will not begin computation until the master is finished) use vision information from the previous cycle, out of step with the world state. In the latter case, the SEI must indicate it is ready for a new cycle immediately after receiving the initial world state, since it would have no earlier vision information.

Connection to the simulation is made in **StartSeI()** of **commun.c**.

Related Protocol Commands: RequestID, AttachID, IAmYourMaster, SetOrganismBody, SetOrganismName, OrganismStarted, TerminateOrganism, ShutDownWorld, OrganismExit, OkToProceed

2.2 Receiving World Changes

Once connected to the simulation, the SEI will be sent a complete copy of the world state; in subsequent cycles, it will be sent only the changes which have occurred in the world, and is responsible for using these to maintain a current copy of the world state. After receiving the updated world state for a cycle, including vision information if being used, the SEI may ignore the rest of the system until its organism has completed its processing, at which time it must indicate that it is ready for the next cycle, and wait for the next set of world changes. The master sends no changes unless it has already verified that they are valid, so the SEI should not need to perform error checking when receiving changes.

The object data structure maintained by the SEI is not identical to that in the master. In particular, where object transforms in the master are relative to higher level objects, all transforms in the SEI are absolute. This will be more efficient only if transforms do not change often; it is done so that an SEI may more easily handle direct references to primitive objects by its organism. **ComputeGlobalCoords()** in **commun.c** does this.

Related Protocol Commands: SetTimeStep, SetWorldTime, SetContacts, SetAtmosphericViscosity, SetDirectedLight, SetAmbientLight, SetAmbientTemperature, CreateObject, SetObject, InsertNextObject, RemoveNextObject, RemoveSubObject, DeleteObject, CreateEmission, InsertNextEmission, RemoveNextEmission, DeleteEmission, StartCycle, StartPerqUpdate, EndPerqUpdate, OkToProceed

2.3 Sending World Changes

At any time after receiving world changes and before indicating readiness for a new cycle, an SEI may request a change to the world as specified by an effect of its organism. It should not make such changes to its copy of the world directly; they will be returned with the next set of changes after being verified by the world master. Before sending a message containing changes, an SEI must request rights and wait for them to be granted; it should hold its rights for as brief an interval as possible.

All changes sent by the SEI are handled in **effect.c**.

Related Protocol Commands: RequestRights, GrantRights, ReleaseRights, Acknowledge, SetObject, CreateEmission, OkToProceed

2.4 World States

Periodically, the master may request that the SEI save its state. It will also inform the SEI of where the master state is saved. The SEI should save its state and respond with a shell command which would cause the SEI, and organism if possible, to be restarted and connected to an existing simulation having the same state as at the time of the save. Such a shell command might be the execution of a saved lisp image, or the execution of the SEI with an argument indicating where a state was written.

The master may also request that the SEI restore itself to a particular state. In most cases this will not be possible, and the master will have to execute the corresponding shell command; if the state of the organism was written to a file, however, the SEI may be able to restore its state without the necessity of restarting it as a new process.

The actual process of saving a state is handled by code specific to a particular SEI and organism.

Related Protocol Commands: SetCheckpointPath, SaveState, RestoreState, RestoreCompleted, UnableToRestore

2.5 Miscellaneous

An SEI may send most of the commands available to a User Interface, if for some reason this needs to be done.

An SEI may receive a broadcast message from the master at any time (these may typically be ignored).

If a message the SEI sends to the master contains an error, the master will indicate as much.

Related Protocol Commands: DisplayMessage, ProtocolError

2.6 Protocol Modification in the SEI

Low-level manipulation of messages is done in **link.c**. Processing of commands received from any source is done in **GetWorldChanges()** and **ProcessMsg()** of **commun.c**, and the routines they call. As mentioned above, changes are produced in **effect.c**. See the code for details.

3. Communicating with an Organism

See *Building Organisms* for specifications of what capabilities a general SEI is expected to provide for organisms; see code referred to for details of how this is done.

3.1 Connection

The core of the SEI is contained in **commun.c**, **link.c**, **effect.c**, and **perceive.c**. For a particular langauge, two additional modules have to be defined: one in the given language, to build appropriate data structures and make available the routines in **effect.c** and **perceive.c**, and one written in C, to provide certain definitions, plus any general functions required to link C to the language. Required definitions are **CKPfile**, a string in which the name of a state file will be written; **OpenDebug()**, a routine accepting a string and opening a debug file if appropriate; **DumpState()**, a routine to save the state of the SEI and organism, in the file named by **CKPfile**; and **ProcessOrgReq()**, which should be a stub unless the organism is a seperate process connected to the SEI by ipc.

3.1.1 FranzLisp

An organism written in FranzLisp will be part of the same process as its SEI. **franzsei.l** provides the interface. The C portion of the SEI is loaded in here, with any C routine to be called directly from lisp explicitly addressed. In addition, the function **(fnaddr)** is defined and its address passed to C routines, enabling C code to call compiled lisp functions when necessary; this linkage is done in **franzlink.c**.

Because of the way in which FranzLisp loads files, all external references in C modules used by FranzLisp must be strictly ordered. This order is utility modules, **link.c**, **franzlink.c**, **commun.c**, **effect.c**, and **perceive.c**; no module higher in the list may refer to objects in a lower module.

The program **buildsei** may be used to create an image of a lisp process with the SEI already loaded in (loading in an SEI takes a long time). The file it creates is very large (a megabyte). This is very useful for an SEI which is stable. **buildsei** requires two arguments: the file containing the SEI lisp code, and the executable file to create. The user will be prompted for these if they are not supplied on the command line. The standard SEI for franzlisp is created automatically by the makefile.

3.1.2 SpiceLisp

A organism written in SpiceLisp must load the module **seiface.slisp**, which is linked to the SEI by a level of ipc beyond the standard protocol of the World Modeling System. Such requests are handled in **ProcessOrgReq()** of **rklink.c**. Currently, only one format for perception is available, and only a limited set of effector functions; additional use of the capabilities of the standard SEI is easily achieved by expanding the protocol in these two modules.

3.1.3 C

An SEI written entirely in C should use a modified version of **perceive.c** rather than an interface to it.

3.2 Perception

World data structures are maintained in **commun.c**. Routines in **perceive.c** are intended to extract fields of these structures observable by an organism, convert information to a specified format (e.g., organism-centered spherical coordinates rather than absolute cartesian coordinates), and store requested information in a buffer so that a module of an SEI written in the language of the organism can build structures appropriate to that language.

Code in such a specialized module of the SEI must directly correspond to code in **perceive.c**; i.e., information must be stored in and taken from the buffer in the same order, and, in the case of vision, the order in which objects appear in the buffer due to successive calls to **nextObject()** implies the tree structure of the world. This is reset by a call to **updateObjectList()**. Touch is buffered in **loadContacts()**. Other senses are not yet implemented.

Functions to return sensory information, such as those described in *Building Organisms*, must be written in the language of the organism, and must call the routines mentioned, in addition to others to extract information from the buffers created. In FranzLisp, C routines **getdouble()** and **getinteger()** retrieve values from the buffer; in SpiceLisp, an entire buffer is transferred in a message and unpacked by similar functions in lisp.

3.3 Effects

Arguments to effector functions must be pointers if the functions are to be called directly from FranzLisp. Changes made should be buffered with the **QueueXXX()** routines provided in **effect.c**; they will be sent after **oktoproceed()** is called. The macro **IFQFULLFLUSH** must be included before queuing a command; **QueueSetSmd()** does this for objects. **WorldCoord()** will convert a vector from organism-centered to world coordinates. See the code for examples.